A Declarative Rendering Model for Multiclass Density Maps

Jaemin Jo, Frédéric Vernier, Pierre Dragicevic, Jean-Daniel Fekete

A Declarative Rendering Model for Multiclass Density Maps

Jaemin Jo, Frédéric Vernier, Pierre Dragicevic, and Jean-Daniel Fekete, *Senior Member, IEEE*

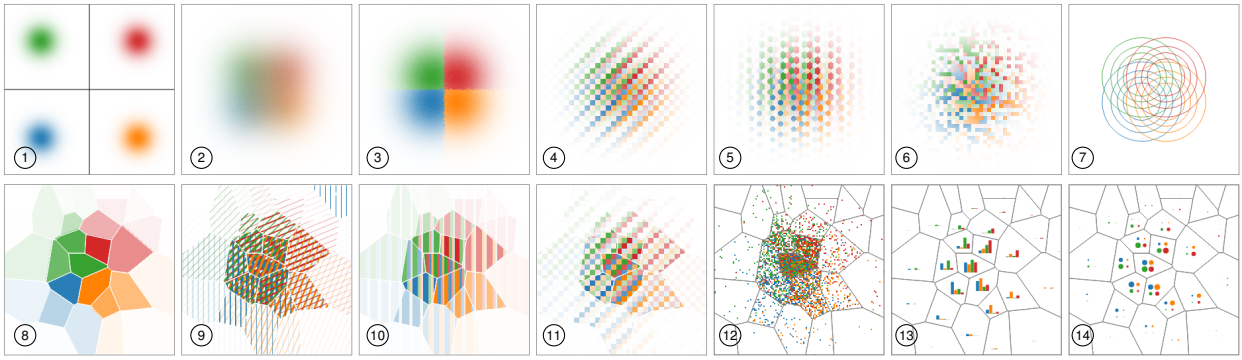


Fig. 1: Design alternatives for a four-class density map. ① shows small multiples where each density map is individually presented with a unique color; ② stacks the density maps and blends the color at each pixel; ③ shows the color of the pixel with the highest density; ④-⑥ use regular and irregular weaving patterns; ⑦ shows a contour plot for each class; and ⑧–⑭ use rebinning (binning and aggregation over the density maps) with tiles produced by a random Voronoï tessellation. The aggregated values are rendered in ⑧ with a flat color showing the highest density, ⑨ with hatching, ⑩ with proportional bars, ⑪ with regular weaving, ⑫ with a dot density plot, ⑬ with bar-chart glyphs, and ⑭ with circle sizes.

Abstract—Multiclass maps are scatterplots, multidimensional projections, or thematic geographic maps where data points have a categorical attribute in addition to two quantitative attributes. This categorical attribute is often rendered using shape or color, which does not scale when overplotting occurs. When the number of data points increases, multiclass maps must resort to data aggregation to remain readable. We present *multiclass density maps*: multiple 2D histograms computed for each of the category values. Multiclass density maps are meant as a building block to improve the expressiveness and scalability of multiclass map visualization. In this article, we first present a short survey of aggregated multiclass maps, mainly from cartography. We then introduce a declarative model—a simple yet expressive JSON grammar associated with visual semantics—that specifies a wide design space of visualizations for multiclass density maps. Our declarative model is expressive and can be efficiently implemented in visualization front-ends such as modern web browsers. Furthermore, it can be reconfigured dynamically to support data exploration tasks without recomputing the raw data. Finally, we demonstrate how our model can be used to reproduce examples from the past and support exploring data at scale.

Index Terms—Scalability, multiclass scatterplots, density maps, aggregation, declarative specification, visualization grammar

1 Introduction

In this article, we are interested in methods to increase the scalability and expressiveness of *2D multiclass maps* (i.e., visual representations of data that consist of two quantitative attributes, which are mapped to (x, y), and one categorical attribute). 2D multiclass maps include scatterplots, multidimensional projections, and thematic geographic maps, altogether called *maps*. These maps are supported by all the multidimensional data visualization and cartographic systems, attesting their popularity and effectiveness. In nonaggregated maps, the categorical attribute is depicted using a categorical visual variable at each point, such as color or shape. However, when the number of points increases, the maps become unreadable because of excessive overplotting, which can result from structural properties of the data (e.g., multiple points being heavily clustered), or simply because of the sheer number of points.

- *Jaemin Jo is with with Seoul National University, Republic of Korea E-mail: jmjo@hcil.snu.ac.kr*
- *Frédéric Vernier is with LIMSI, CNRS, Univ. Paris-Sud, Université Paris-Saclay. E-mail: frederic.vernier@limsi.fr*
- *Pierre Dragicevic and Jean-Daniel Fekete are with Inria E-mail: firstname.lastname@inria.fr*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

Massive datasets suitable to be visualized as multiclass maps are easily available, for example, the *RTI U.S. Synthetic Household Population*™ [50] containing one point per person in the United States (300 million) with their age, sex, race, income, and house location. Large multiclass maps can also be easily generated by computing the projection of millions of multidimensional multiclass points using modern scalable projection systems [39, 48].

To scale scatterplots, several approaches have been proposed, such as adaptive opacity [15, 30, 32] and aggregation [13, 53]. However, adaptive opacity does not scale well with the number of categories since multiple categorical colors become ambiguous when blended, and aggregation methods such as density plots are limited to purely bivariate quantitative data. Few techniques have been described to support the visualization of aggregated multiclass maps, and to our knowledge, no system supports their visualization in a flexible way.

In this article, we present a declarative model to specify *multiclass density maps*, multiple density plots with different classes, applicable to an arbitrary number of points. Our contributions are:

- a review of visualization techniques for multiclass density maps,
- a conceptual model for describing a wide range of visualizations of multiclass density maps, and
- a concise declarative grammar and its interpreter to specify their rendering.

Our model relies on the creation of multiple aggregated *data buffers* by visualization library back-ends, while a front-end system (e.g., a web browser) allows interactively configuring and combining the data

buffers to create visualizations. The model enables many types of interactive reconfigurations of the rendering pipeline, thanks to the speed of the modern front-ends, without reviewing the raw data. Providing a clean conceptual model supporting efficient implementations will improve the exploration of multiclass maps at scale. It will also allow visualization systems to implement an effective separation in their pipeline between data aggregation and the rendering of aggregated data for greater richness and scalability of multiclass data visualization.

2 Related Work

Several articles [13, 23, 31, 53] address the problem of scaling the visualization of multiclass maps. We identified three facets of scalability:

- *data size* related to the number of data points and categories,
- *perceptual processing* related to the ability to perform some tasks efficiently given a *data size*, and
- *computation speed* related to the time to compute an image from a visualization technique given a *data size*.

Ideally, we want visualization techniques to support large data sizes (100 million points or more with tens of categories or more) while allowing performing important perceptual tasks quickly, and preferably with a high computational speed to allow interactive exploration, implying a refresh rate between 25 ms to 10 s, depending on the type of interaction [34, 44]. However, managing large data sizes while allowing effective perceptual processing is already a challenge, and our goal is to design a model that can improve all the three facets of scalability.

2.1 Tasks

Multiclass maps, such as scatterplots, multidimensional projections, and thematic geographic maps, are used to perform several important tasks, reviewed in multiple articles. Munzner presents a generic framework for tasks [35] where she distinguishes *actions* and *targets*. She lists three categories of actions useful on most visualizations and four categories of targets. Her framework can be applied to every visualization technique, but the list of actions and targets can also be specialized, depending on the particular visualization techniques and application domains. Specializing Munzner's framework for scatterplots, Sarikaya and Gleicher [42] define a taxonomy with 12 tasks, sorted in three categories: *object-centric*, *browsing*, and *aggregate-level*.

For maps, Roth [41] also adds specific actions and targets such as rank, associate, delineate, reexpress (changing the assignment of visual properties and layout), arrange, sequence, resymbolize, overlay, and reproject. For dimensionality reduction, the literature is more dispersed. Most of the articles mention visualizing class separation and class quality (e.g., [3]), but more elaborate studies such as Brehmer et al. [8] distinguish two categories of task sequences: *Dimension-Oriented* and *Cluster-Oriented*. The cluster-oriented tasks are specifically related to multiclass maps. Aupetit [2] mentioned more subtle tasks related to distinguishing data outliers vs. projection artifacts; they imply visual relations between objects and clusters. In addition, many tasks related to multiclass maps involve comparisons, mentioned in tasks 10–12 of Sarikaya and Gleicher [42] and also studied by Gleicher [18].

To sum up, when the data size increases, visualizations cannot clearly show all the individual points. Therefore, tasks related to objects/points become impossible unless a small and useful subset of important points can be selected and clearly visualized. In addition, comparison tasks can be performed over the cross product of objects or groups, and related to values, locations, distributions, classes, and other meaningful concepts. Therefore, it is unlikely that one visualization technique will allow performing all the possible comparisons efficiently.

2.2 Nonaggregate Methods

When the amount of overplotting is limited, several effective techniques have been introduced, such as jittering [24] or using alpha blending, sometimes with adaptive opacity [15, 30, 32]. Jittering requires empty areas to spread crowded points; therefore, it does not scale when overplotting happens on large areas. For alpha blending, when multiple colored points are drawn on top of each other using transparency, their colors are blended, and it becomes difficult to even perceive if one category is present at all, except for the specific case of two or three classes with a well-chosen colormap [29, 49].

Chen et al. [12] have recently introduced animation to alleviate overdraw in multiclass scatterplot matrices (SPLOMs). They animate the drawing order of the points to help the user see every point at any position by watching it for a sufficient time. However, that technique is also limited by the time to animate the whole dataset. The authors mention an animation speed up to 800 rows per second, which would take hours or days to run through millions of points. Even if the user was willing to spend time watching the animation, previous studies have shown that visual statistics can only be done over short periods [1].

When the number of data points increases, a seemingly simple solution consists in *sampling* points to limit overplotting. While uniform sampling is easy and fast to perform, it produces artifacts that are difficult to interpret cognitively [5, 28]. If density significantly varies (e.g., population maps), some regions of the map become empty, while others remain crowded. Thus, many areas are hard to analyze. More complex sampling methods have been studied; they are still computationally expensive [5] and produce artifacts that are challenging to interpret.

2.3 Aggregate Methods

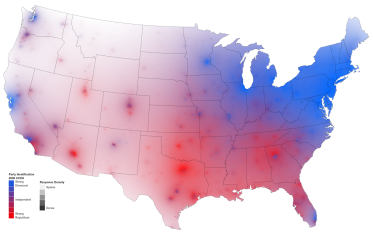
For situations where a few regions are overplotted but others are sparse or empty, Mayora and Gleicher have designed Splatterplots [31] that replace dense regions with covering polygons, show individual points in sparse regions, and increase the visibility of outliers. Still, when the number of categories and the overall density increase, Splatterplots degrade since filled contours cover most of the visualization.

When the number of points significantly exceeds the number of pixels, it becomes technically impossible to distinguish the individual points, to perceive their density, or to see what categories are present at any pixel. All the techniques described before fall short: jittering cannot be used because there is no extra space to spread the high-density areas; alpha blending is limited to a few layers of overplotting and, even then, does not allow distinguishing clearly what colors have been mixed; and Splatterplots cover the whole display with filled contours.

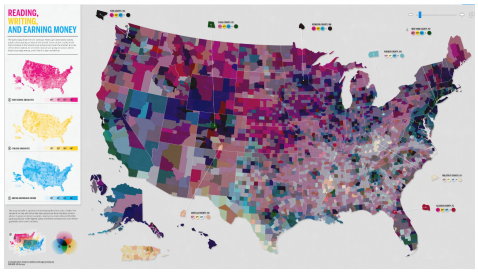
To visualize maps of larger datasets, a few models have been proposed. *The Grammar of Graphics* [54] provides a general model for visualization where scalability is addressed through "statistical methods," including *binning* with *summary statistics* and *smoothing*. However, it does not specifically address the problem of multiclass maps. Wickham [53] describes a framework to achieve scalability called *bin-summarise-smooth* (BSS). It starts from a dataset with two quantitative variables, bins them, computes a simple aggregation statistics on the bins such as *count*, and then applies a smoothing function before rendering the results on the screen. Wickham insists on many important details that need to come with this pipeline, such as the management of outliers and uncertainties. However, his framework is limited to bivariate data, not multiclass.

Cottam et al. describe a similar framework called *Abstract Rendering* [13, 23] (AR) made of four stages: *select*, *info*, *aggregate*, and *transfer*. The *select* function is equivalent to the *bin* stage of BSS, but in a specific case where each bin is rectangular and aligned with screen pixels. While BSS is based on standard aggregation statistics where a bin contains a single value, AR extends the concept of bin to compound values, including RGB colors, list of categories, or reference to programming objects; these values are extracted from the data with the *info* function. Therefore, the *aggregate* functions are more diverse than the *summarise* step in BSS.

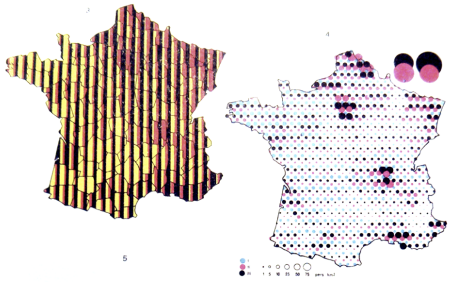
Various visualization methods have been used to visualize multiclass maps. One of the most widely used methods is color mixing where each class is assigned a unique color, and the colors are mixed at each point of a map by mixing depending on the density as in Fig. 2a. Color mixing can be extended to more classes; for example, Slingsby et al. [45] uses seven unique colors to represent different classes. *Contour plots* can be used to visualize multiclass data when the data is smooth enough to avoid too many tangled lines [31]. Ware introduces *Texton maps* [52] where "texton shapes" are overlaid on a colored density map to show a second class, although with a limited spatial resolution for the texton shapes. Miller introduces *Attribute Blocks* [33] where multiple



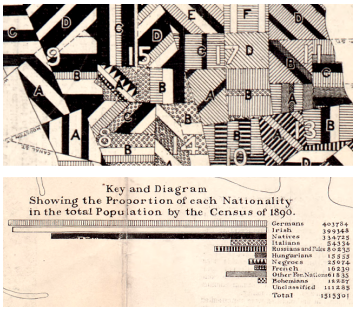
(a) Support for Democratic vs. Republican candidates in 2008 [46].



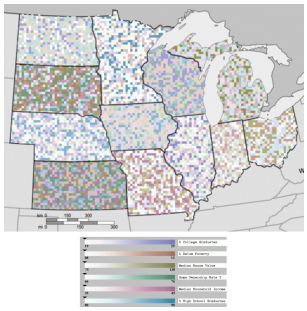
(b) Percentage of high school graduates, of college graduates, and median house income in 2009 [22].



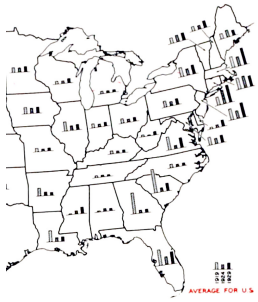
(c) Number of workers per sector of economy (primary, secondary, tertiary) in 1954 by Bertin [4].



(d) Detail of a map of New-York City showing the distribution of nationalities across districts in 1890 [25, 40].



(e) Six socioeconomic indicators in each of the twelve Midwestern US states [19].



(f) Detail of a map showing average sales per farm for each US state in 1919, 1924, and 1929 [10].

Fig. 2: Examples of multiclass density maps. Various techniques have been used to visualize multiclass density maps, such as bivariate (a) and trivariate (b) maps, exhaustive maps [4] (c), hatching (d), weaving (e), and glyphs (f).

density maps are computed, one per class, and assigned a categorical color (a hue). The visualization space is then segmented by a grid with each cell showing one of the density maps. Similar to Attribute Blocks, *Weaving* [19] has been designed to visualize multiclass data over choropleth maps (Fig. 2e). One colored choropleth map is created for each class, and the final image is composed by stacking the choropleth maps using a regular grid where each cell shows a density map of a randomly chosen class. Superimposing symbols (or, *glyphs*) on a map is also commonly used to visualize multiclass data in cartography (Fig. 2f). For example, Brewer and Compbell [9] introduce varied point symbols, such as a pie chart with two wedges, to visualize bivariate data on maps. As follow up studies, Nelson evaluates the performance of those symbols [36], and Lamb [26] presents a layout algorithm that automatically removes the overlap between the symbols. Finally, *dot distribution maps* (or *dot density maps*) [17] look similar to sampling, but they generate a random uniform pattern over aggregated areas to convey density (Fig. 1-⑫). Each dot represents a constant number of data points, and the user needs to remember that meaning.

To summarize, various techniques have been proposed in the literature and visualization packages for multiclass maps. However, to deal with the large number of tasks the user may want to do with this type of maps, having a unifying conceptual model that can describe those methods and realize new promising methods is required.

3 Examples from Cartography

In this section, we discuss a few representative examples of multiclass density maps, shown in Fig. 2. Our visualizations are taken from cartography, because this significant discipline often needed to visualize multiclass density data, so it has rich examples. However, as we discussed before, multiclass density maps do not need to be cartographic. For example, they can include the combinations of multiple scatterplots showing abstract data [12].

The first example by David B. Sparks [46] shows data from the 2008 Cooperative Congressional Election Study (CCES), where 30,000 randomly-sampled US residents were asked to report their support for Democratic vs. Republican candidates (Fig. 2a). Areas with strong Democratic and Republican support are in blue and red, respectively. In addition, colors in densely populated areas are highly saturated, while areas with low population density appear "washed out." While the divergent red-blue scale does not encode density information (but a mean response on a seven-point scale), this map can alternatively be seen as a mixture of two density maps: one showing the density of Democratic supporters (and encoded on a white-blue scale) and one showing the density of Republican supporters (and encoded on a white-red scale). Therefore, this map can be conceptually seen as a multiclass density map with two classes.

The second example is a US map by Gregory Hubacek [22], showing the percentage of high school graduates (in magenta), college graduates (in yellow), and the median house income (in cyan) in 2009. The legend uses small multiples, a simple and common technique for multiclass density maps. The large map combines the three maps using subtractive color blending (e.g., as when stacking colored filters on a white surface). Regions that are high in all three indicators are black, while regions that are high in only one or two of the indicators have a recognizable color. Even though the third class (median house income) does not involve density data, the same technique can in principle be used to produce pure multiclass density maps with three classes.

The two maps of France in Fig. 2c are obtained from Jacques Bertin's book *Semiology of Graphics* [4]. Both maps show the number of French workers per geographic department in 1954, broken down into three economic sectors: the primary (agriculture), the secondary (industry), and the tertiary (commerce, transports, services). In the left map, the sectors are encoded in yellow (I), red (II), and black (III). In the right map, they are encoded in cyan (I), magenta (II), and black (III). While the left map only shows proportions, the right map shows absolute counts. Bertin called these maps "exhaustive maps" [4]. Following our terminology, they are multiclass density maps with three classes. These designs provide a good overview of the data, and likely better support local comparisons and value retrieval than the two previous designs.

Fig. 2d illustrates an even older example of a multiclass density

map. It is a portion of a map showing the distribution of nationalities in New-York City in 1890 [25, 40]. Each nationality is encoded with a specific texture, as explained in the legend underneath. Each sanitary district is filled with textured bands whose widths map to the proportion of nationalities within the district. Thus, districts with less diversity (e.g., 17^{th}D) appear visually homogeneous, while districts with more diversity (e.g., 15^{th}A) exhibit complex patterns. Although the encoding may appear overwhelming, this multiclass density map manages to show as many as ten classes, without the use of color. It is analogous to the left map from Bertin in Fig. 2c, except the orientation of bands is varied across districts to make boundaries more prominent.

The choropleth map in Fig. 2e shows six socioeconomic indicators measured in each of the twelve Midwestern US states [19]. Each indicator has a color scale, as shown by the legend underneath. Each state blends the six color scales using the recently introduced *color weaving* technique [19], where colors are displayed side by side in a high-frequency texture. The state of Kansas (on the bottom left) appears dark because it is high in all six indicators, while the state of Minnesota (second on the top) is high in the number of high-school graduates (dark cyan dots) but relatively low in other indicators. A study has suggested that color weaving facilitates value retrieval from individual channels compared with color blending, as used in Fig. 2b [19].

Fig. 2f is an example of what Robert L. Harris calls "symbols on maps" in his illustrative reference on information graphics [20]. This map shows the average sales per farm through cooperative associations for each of the US States in 1919, 1924, and 1929 [10]. If the quantity shown was density information (e.g., population density), this map would be a multiclass density map with three classes (i.e., one class for each year). In this article, we will use the term "glyph" to refer to such miniature visualizations. In his book *Semiology of Graphics* [4], Bertin advises against using glyph maps because they typically provide poor overviews and make it hard to perceptually separate the channels (compare with Fig. 2c, for example). However, Bertin only focuses on showing a few channels (typically three), while visualizations such as in Fig. 2f can be easily extended to many more channels.

4 The Class Buffer Model

In this section, we introduce a declarative model that covers a large design space of multiclass density maps. It originated from our collection of real examples of multiclass density maps from cartography, information visualization, and infographics, such as the ones described in the previous section. Reviewing the examples, we found that most examples could be expressed with the *Class Buffer idiom*, particularly used in a previous study [12]; we use that idiom as the building block for our model.

Our implementation of the model defines an expressive visualization grammar for multiclass density maps using the JSON data-interchange format. Such specifications can be efficiently interpreted and rendered in visualization front-ends such as modern web browsers. Our model consists of six stages (see Fig. 3):

1. **Binning**. The initial data table is split into as many *data buffers* as classes. A data buffer is essentially a 2D histogram that counts and bins all data cases of a particular class. This step is performed by the back-end of the visualization system, which sends the data buffers to the front-end visualization engine.

2. **Preprocessing**. Once in the front-end, the data buffers undergo optional preprocessing operations such as Gaussian smoothing.

3. **Styling**: The data buffers are then transformed into *class buffers*. A class buffer is a data buffer augmented with visual properties, such as a class-specific color.

4. **Rebinning**. The grid used by class buffers is partitioned into *tiles*, and for each tile, an *aggregated count* is computed for each class and stored in a *data vector*. Aggregated counts are then converted into *normalized counts*.

5. **Assembly**. The assembly stage turns tiles and normalized counts into a single *density map image*. Several types of assembly operations are possible, namely, *masking*, *mixing*, *hatching*, and *generating glyphs*.

6. **Rendering**. This final stage renders the density map image on top of a background image, also adding decorations and annotations (landmarks, axis, legend, etc.) to make the visualization easier to read and interpret.

Each of the six stages is described in more detail below. We assume we start from a tabular dataset T with three attributes (Q_1, Q_2, C), with Q_1 and Q_2 being two quantitative attributes and C being a categorical attribute with N possible **classes** $C = \{c_1, c_2, \cdots, c_N\}$. We further assume that (Q_1, Q_2) use a 2D position encoding (i.e., Q_1 is mapped to the x axis and Q_2 is mapped to the y axis in the final visualization).

4.1 Binning

The Class Buffer model is designed to scale multiclass scatterplots while supporting responsive interactions at the user interface level. To this end, we start by preprocessing the raw data so that multiclass density maps can be later rendered in interactive time.

In the binning stage (Fig. 3-1), the initial data table $T = (Q_1, Q_2, C)$ is split into N **data buffers** $B_{i \in [1 \cdots N]}$. A data buffer is a $W \times H$ scalar matrix that discretizes the domain of (Q_1, Q_2) into a regular grid and stores the number of data cases that fall into each cell (which we refer to as **counts**). Data buffers are class-specific, so one data buffer is created for each of the N classes in C. In other words, each data buffer B_i stores a 2D histogram for the subset of the data that verifies $C = i$.

We refer to a cell in a data buffer as a **binned pixel**. The granularity of the binning (i.e., the choice of values for W and H) typically depends on the required rendering resolution for the final visualization. Naturally, larger values for W and H allow for a higher rendering quality at the cost of bigger buffer sizes.

In a Class Buffer model implementation, the binning stage is expected to occur offline on a back-end server or to use precomputed tiles for fast rendering [27]. This stage can also be progressively carried out [16]. Since binning is independently done for each class, it can be very efficiently parallelized with modern hardware such as GPUs and multicore CPUs. In addition, the data buffers can be compressed before they are sent to a browser; by default, our implementation sends 16-bits deep grayscale PNG images that are supported by all web browsers, and where each pixel corresponds to a binned pixel in a data buffer. The size of the data buffers can be adapted to the network bandwidth, screen resolution, and processing power of the machine managing the front-end visualization and interaction. All computation and rendition at the later stages occur on the front-end.

Note that the set of data buffers can also be seen as a 2D N-variate dataset where each of the $W \times H$ cells has N density values, one per class. In cartography, maps are often referred with their number of variates; for example, Fig. 2b is a trivariate map (i.e., three classes).

4.2 Preprocessing

The preprocessing stage applies optional transformations to the N data buffers. Possible transformations are filtering out classes (e.g., ignoring the data buffer of the "mouse" class as in Fig. 3-2), combining two or multiple buffers into a single data buffer, and normalizing where the counts in each data buffer are divided by the sum of the counts. Our current implementation supports smoothing (i.e., the application of a Gaussian kernel of arbitrary size) (see Fig. 6a for an example). According to Wickham, "Smoothing is an important step because it allows us to resolve problems with excessive variability in the summaries" [53]. The outcome of this process still consists of N data buffers, but the counts can possibly take noninteger values.

4.3 Styling

The styling stage (Fig. 3-3) constructs N **class buffers**, one per class. A class buffer is a data buffer that has been assigned visual properties that will be used in the later stages of the rendering pipeline. These visual properties include, for example:

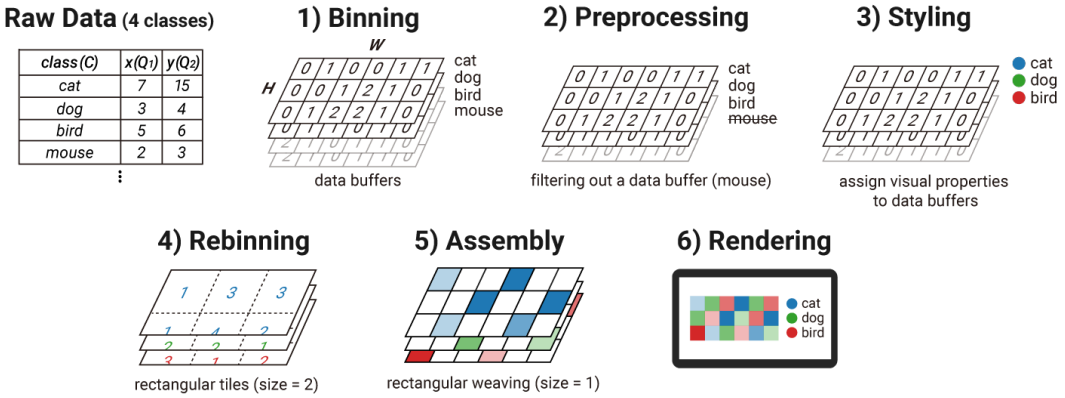


Fig. 3: The six stages of the Class Buffer model.

- a *color* that will determine the color scale used to visually represent the class (see Fig. 1);
- a hatching *angle*, in case hatching is used (see Fig. 1-⑨);
- a *scale* that will be used to transform the counts before encoding them visually (e.g., linear scale, log scale, or square-root scale).

All these visual properties are specified on the front-end using a grammar that will be detailed in Sect. 4.8. All the visual properties have default values. For example, by default, we assign a color from ColorBrewer's qualitative color scales [21] to each class buffer. However, controlling the color assignment is often important, as in Fig. 2-a where the parties have well-known colors. Besides channel-specific visual properties, this declarative grammar contains information on data buffers (e.g., a URI to data buffers on a server) and specifies global options for the pipeline (e.g., how color scales should be blended or how tiles should be defined for the next stage).

4.4 Rebinning

The rebinning stage (Fig. 3-4) first partitions the $W \times H$ grid into M **tiles**. A tile is a set of cells in the $W \times H$ grid, and the set of all tiles is referred to as a **tiling**. Tilings are defined such that there is no overlap between tiles and the union of all tiles is the whole $W \times H$ grid. The simplest tiling is **pixel tiling**, where each tile consists of a single cell of the $W \times H$ grid (i.e., $M = W \times H$). Other tilings contain tiles that span multiple cells. Such tilings can be either regular (e.g., a set of 2×2 rectangles as in Fig. 3-4) or irregular (e.g., regions in a choropleth map). Fig. 1-⑧–⑭ are examples of irregular tilings, while Fig. 1-①–③ use pixel tiling. Tiles can be defined based on geometrical primitives or using the URI of a TopoJSON [6] specification to define geographic administrative boundaries for choropleth maps.

After a tiling has been constructed, the rebinning process creates and assigns a **data vector** to each of the M tiles. A data vector is a vector of length N that stores a count for each class, called the **aggregated count**. That is, all binned pixels from class i that belong to the tile t are aggregated into a single value and stored in the i^{th} element of the data vector of t. Possible aggregate functions include *sum* (i.e., the counts of the binned pixels are summed), *min*, *mean*, *max*, and *density* (sum divided by area). Again, counts can take noninteger values. When pixel tiling is used, aggregation amounts to simply copying counts from binned pixels into data vectors.

Finally, the rebinning stage virtually normalizes all aggregated counts between 0 and 1, according to the options assigned during the styling stage (e.g., using a linear, log, square root, or equi-depth histogram scale). These counts are called **normalized counts**. Concretely, it defines a scale object that maps the range of counts to $[0, 1]$. This scale object is later reused for the legend.

4.5 Assembly

The assembly stage (Fig. 3-5) turns the tiles and data vectors into an image with an opacity (alpha) channel, which we refer to as a **density map image**. Much of the visualization process occurs in this stage. There are four broad types of assembly operations: masking, mixing, hatching, and generating glyphs.

The first type of assembly operation, **masking**, assigns a mask to each of the N class buffers. A mask is a $W \times H$ grid of opacity values, and the N masks are defined so that the sum of opacity values across all classes is 1 for each of the pixels. Then each tile is rendered N times with a uniform color. The color corresponds to the color previously assigned to the class in the styling stage, and its opacity is set to the normalized count stored in the data vector. The N tiles are finally alpha-blended using the opacity values stored in the masks. Masks are essentially used for producing **weaving** patterns, such as illustrated in Fig. 1-④–⑥. In these examples, the mask opacity values are either 0 or 1, and the masks are used to define "patches." Patches can be polygons, such as triangles, rectangles (Fig. 1-④), or hexagons (Fig. 1-⑤). They can be randomly assigned to class buffers (Fig. 1-⑥) or follow a regular pattern (Fig. 1-④). Weaving guarantees that each pixel in the density map image is assigned a unique class. When no masking is specified, conflicts can be resolved using other types of assembly operations.

The **mixing** operation combines tiles by blending them. Similar to masking, each tile is rendered N times, again with a uniform color corresponding to the class and an opacity value equal to the normalized count. However, the colors are mixed across the entire tile instead of being masked. A straightforward mixing approach consists in *averaging* all colors (see Fig. 1-② for an example). Other mixing methods can be used; for example, *additive mixing* sums each RGB channel of the N colors, thus generating bright colors for high-density regions. *Multiplicative mixing* does the opposite, yielding dark colors for high-density regions (see Fig. 6b for an example). It is also possible to take a *winner-takes-all* approach, where only the class with the highest count is chosen (see Fig. 1-⑧ for an example). Finally, we also define a *loser-takes-all* approach, where only the class with the lowest nonzero count is chosen, but its color intensity is inverted; low color values become vivid. This mixing method can boost the visibility of outliers.

Tiles can also be rendered with a **hatching** operation, which fills each tile with evenly spaced lines. Typically, normalized count is encoded with line thickness, while class is encoded using line orientation, color, or texture. The hatches can be combined across classes either by stacking them side by side within each tile (as in Fig. 2c-left) or by superimposing them (as in Fig. 1-⑩).

Finally, tiles can be rendered using a more conventional visualization pipeline, i.e., by **generating glyphs**. Glyphs are miniature visualizations that encode all N normalized counts in the data vector and are typically displayed at the center of each tile. For example, in Fig. 1-⑬, a bar chart is placed inside each tile, conveying the density for each class. Meanwhile, Fig. 1-⑭ uses a "punchcard" style, where densities are mapped to circle radii. In our implementation, all such glyphs can be specified in Vega-Lite [43], a high-level JSON grammar for visualization. Note that finding a "good" location for the glyphs is not always trivial. We currently compute the largest rectangle in polygon for each tile and place the glyph at the center of that rectangle.

The assembly operation is optional. When no assembly operation has been specified, each tile is rendered using a uniform translucent color as previously described, and the process outputs N density map images instead of a single one, leaving the conflict resolution to the final rendering stage.

4.6 Rendering

The final rendering stage turns the density map image(s) into a final image that can be directly displayed on the screen (Fig. 3-6). In case the classes were already assembled in the previous stage, the density map image is simply rendered on a background. The background color defines the lower end of the color scales used in the final visualization. For example, a white background produces color scales where zero or minimum density is mapped to white (as in Fig. 1). While backgrounds can be uniform, they can also consist of cartographic backgrounds that provide extra annotations such as city locations and names. Alternatively, annotations can be rendered on top of the density map image. The rendering stage augments the rendered density maps with any extra visual element necessary to improve readability and interpretability. This stage is responsible for rendering optional contour plots (see Fig. 1-⑦), tile boundaries, the x and y axes, and the legend (discussed in more detail in the next section). Finally, it decides on where to render the density map image and at what scale (e.g., in case of pixel magnification). In the case of multiple density map images (i.e., no assembly), each density map image is rendered at different locations (i.e., a *small multiples* approach, see Fig. 1-①).

4.7 Legends

To be usable, a multiclass density map needs a legend. Our current implementation of the Class Buffer model automatically generates simple legends (Fig. 4). Our legends consist of three parts: (1) a *key* that maps class colors to class names, (2) a *scale* to help retrieve counts, and (3) an optional *explanation* of how mixing is done.

The *key*, present in all our legends, lists the name and color that is assigned to each class buffer. The *scale* shows how counts map to visual attributes. In most cases, it consists of a set of color ramps (Fig. 4 [a–c]). The color ramps linearly interpolate between the colors of the lowest and the highest count and add ticks that are equally distant in the data domain. Thus, if a nonlinear scale has been specified, the ticks are unevenly spaced in the legend, indicating that the scale is nonlinear (Fig. 4 (b,c)). For glyph-based assemblies, the scale consists of one or several glyphs with numerical labels, such as the miniature bar chart in Fig. 4d or the circle-radius scale in Fig. 4e.

The last *explanation* area illustrates which mixing function (e.g., *max* as in Fig. 4a or *mean* as in Fig. 4b) was used for color mixing. We considered using an Euler diagram or InfoCrystal [47] representation to visualize all possible combinations of class colors. However, in our case, the luminance level of class colors can vary depending on density, and it is hard to simultaneously visualize the mixture of multiple colors at different luminance levels. More generally, visualizing a multivariate color scale with more than two variables is impossible, because it would require the volumetric visualization of a cube or hypercube. Thus, we only show the bivariate color scale derived from the first two class buffers. Although this approach does not support value retrieval, it explains how colors are mixed. We do not provide a similar explanation for techniques such as masking and hatching because the way classes are combined can be deduced by looking at the visualization itself. For example, in Fig. 4c, class buffers are spatially separated through weaving and do not involve any hidden mixing.

These techniques are meant to provide basic support for legends, and can be improved or extended. For example, the key and scale could be combined, as it is commonly the case in visualization and map legends.

4.8 Implementation

Our implementation is available, with example datasets, at `https://github.com/e-/Multiclass-Density-Maps` and examples can also be explored at `https://jaeminjo.github.io/Multiclass-Density-Maps/`.

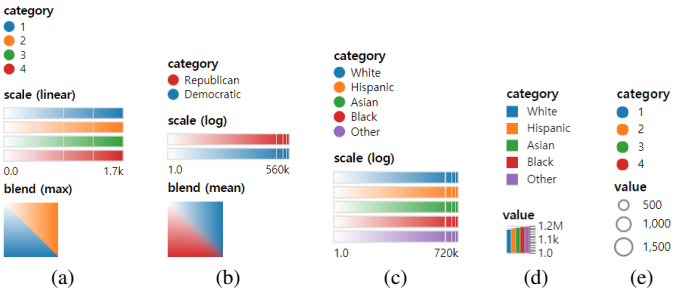


Fig. 4: Auto-generated legends for multiclass density maps

```
{"description"?: <string>,
 "background"?: <Color>,
 "data": {"url": <url> | "dataSpec": <DataSpec>},
 "smooth"?: {"radius": <number>},
 "reencoding"?: {
   "label"?: <LabelSpec>,
   "color"?: <ColorSpec>,
   "hatching"?: <HatchingSpec>},
 "rescale"?: {
   "type": "linear"|"log"|"pow"|"sqrt"|"cbrt"|"equidepth",
   "rebin"?: {
     "type": "none"|"square"|"rect"|"topojson"|"voronoi",
     "aggregation": "mean"|"max"|"sum"|"min"|"density",
     "width"?: <number>, "height"?: <number>,
     "size"?: <number>,  "topojson"?: <TopoJSONSpec>,
     "url"?: <string>, "feature"?: <string>,
     "points"?: <Point[]>, "stroke"?: <Color>},
   "compose"?: {
     "mix": "none"|"invmin"|"mean"|"max"| "blend"|
            "weavingrandom"|"weavingsquare"|"weavinghex"|
            "weavingtri"|"propline"|"hatching"|"separate"|
            "glyph"|"dotdensity"|"time",
     "mixing"?: "additive"|"subtractive"|"multiplicative",
     "size"?: <number>, "widthprop"?: <string|number>,
     "colprop"?:<boolean>, "order"?: <number[]>,
     "glyphSpec"?: <GlyphSpec>, "interval"?: <number>},
   "levels"?: <number>
 },
 "contour"?: {
   "stroke": <number>, "lineWidth"?: <number>,
   "values"?: <number[]>, "blur"?: <number> },
 "legend"?: <LegendSpec>, "stroke"?: <StrokeSpec>,
 "axis"?: <AxisSpec>}
```

Fig. 5: Syntax of Class Buffer specifications

Our Class Buffer model is implemented in approximately 5,000 lines of TypeScript, a strongly typed language that can be transpiled into JavaScript. We render the tile glyphs using Vega-Lite [43], which is conveniently also written in TypeScript. We also rely on the D3 library [7] for contours and cartographic projections.

Interpreting a specification takes between a few hundred milliseconds to one second depending on the complexity of the operations to perform, not counting the time to transfer the data, including data buffers and the TopoJSON file if needed. Currently, data buffers can be sent as 2D arrays in the JSON format or as gray-scale 16-bit PNG files.The data buffers in the JSON format are usually heavily compressed by the gzip compression of the HTTP protocol when enabled, as in our example gallery. Changing a specification and reinterpeting it is usually much faster since data is not transferred, and intermediate operations, such as rebinning, can be cached if the tiling is not changed, which is common for maps.

Modern browsers support three graphic libraries: SVG, Canvas, and WebGL. Our implementation is meant to be easy to read and extend; therefore it uses the Canvas, which is faster than SVG but

less complicated to program and understand than WebGL. For screen resolutions up to FHD (1920×1080), the rendering time is under 0.1 s for most of the configurations and optimizing it is not necessary.

Fig. 5 summarizes the syntax of our visualization specifications. While it may look complex, most specifications in practice are very concise; we show several concrete examples in the next section. The data specification mentioned on line 3 (`<DataSpec>`) comes from a back-end program; it contains URIs to the data buffers and the information required to visualize them, such as the field names, domains, ranges, data-specific colors (e.g., red for Republican and blue for Democrats), and sometimes projection for cartographic data. The rest of the syntax describes the operations to apply to the data buffers, such as smoothing, styling (`reenconding`), rebinning, assembly (`compose`), normalization (`rescale`), and options for the post-processing stage (`contour`, `legend`, `stroke`, and `axis`).

5 Examples from the Class Buffer Model

In this section, we show how the Class Buffer model can be used to create multiclass density maps from data in practice. We start by reproducing our classic examples from cartography (Fig. 2). Then, we use the Class Buffer model to scale up conventional scatterplots, and demonstrate several design alternatives that can be configured interactively on front-ends to support various tasks.

5.1 Revisiting Examples

One strength of the Class Buffer model is its ability to express various designs of multiclass density maps with a unified visualization grammar. Fig. 6 shows the reproduced examples of Fig. 2 with synthetic data. Note that all maps are complemented with legends.

Fig. 6a smooths prebinned histograms with a Gaussian kernel of radius 1. Rebinning is not used in this map, so each prebinned pixel is directly rendered on a canvas by default, with a color determined by a logarithmic color scale. For color mixing, the *mean* blending function is chosen. This blending function outputs an achromatic color if two parties are equivalently supported and a saturated color if one party outperforms the other. Note that Fig. 6a has the boundaries of states through the `stroke` option, which does not affect binning but is handled at the rendering stage.

The other maps apply rebinning on prebinned histograms, which is specified through the `rebin` option. Fig. 6b uses county-level rebinning, while the other US maps perform rebinning at the state level. In addition, Fig. 6b adopts multiplicative color blending, so larger data values from class buffers produce a darker color. Fig. 6c and Fig. 6d employ hatching with aligned and nonaligned lines, respectively. In Fig. 6c, each county is painted with stacked bars that show the proportion of each class. By contrast, Fig. 6d assigns a different angle to each class, encoding a data value using both the color and thickness of a line. Fig. 6e shows an example of random weaving where each class buffer occupies a random subset of rectangular patches on a tile (i.e., a state). Finally, Fig. 6f places tile glyphs, a mini bar chart, on each state.

5.2 Large Multiclass Maps

Multidimensional projection algorithms are now applicable to millions of points in a reasonable time [39, 48]. We applied the LargeVis algorithms [48] to the notMNIST dataset [11] made of approximately 530,000 small images (each has $28 \times 28 = 768$ gray-level pixels). The algorithm uses these 768 values as high-dimensional vectors that are projected in two dimensions; the results are shown in Fig. 7.

The main task analysts perform on these projections is checking that classes are well separated by comparing the visual clusters with the color labels and looking for erroneous points. Fig. 7 shows two configurations of the same Class Buffer that reveal the ability of LargeVis to separate some of the classes on the left and the large number of erroneous points on the right, revealing outliers and algorithm artifacts. The first configuration composes the image by showing the class with the highest density at each pixel and rescaling the color histogram using an equi-depth histogram. This configuration only shows the dominant class at each pixel and depicts a good class separation, except at the center where multiple classes are mixed. By contrast, the right image employs the `invmin` mix method that returns the lowest data value at each pixel. As a result, the right image allows focusing on outliers and artifacts to understand where the algorithm has not been able to separate classes properly.

Fig. 8 visualizes the on-time performance of three carriers in the United States. Each row in the dataset [37] has two quantitative attributes, *travel distance* and *arrival delay*, and one categorical attribute, *carriers*. The original dataset had 2.4 million flights, and we created three data buffers (one for each carrier) by binning the dataset along with the two quantitative attributes. We then normalized the counts to percentages to allow comparison among carriers. Fig. 8 shows the result with two density map designs. Unlike previous examples, the density maps are augmented with the x and y axes. Fig. 8-left uses the *max* mixing function with a three-level equi-depth color scale (i.e., a discrete color scale where every color band has the same number of rows) and reveals the overall trend between distance and arrival delay; most flights traveled less than 2,500 miles, and most long delays occurred from those flights. For short flights (i.e., $distance <$ 1,000 miles), DL had the largest number of delayed flights (orange tiles around the top-left corner), whereas UA had the largest number for intermediate flights (i.e., 1,500 miles $< distance <$ 2,500 miles, green tiles at the center). Fig. 8-right uses parallel proportional lines to display the data as 100% stacked bar charts in each tile. One can see that AA (blue bars) generally performed well for short and intermediate flights but not for long flights (i.e., $distance >$ 3,500 miles).

6 Discussion

In this section, we discuss the benefits of the Class Buffer model, how it can combine with a traditional visualization system, including the postprocessing stage where additional elements should be combined with our model to address important tasks.

6.1 Benefits of the Class Buffer Model

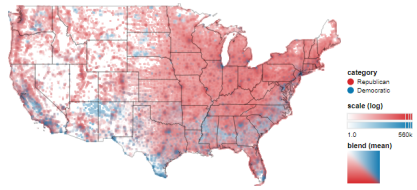
Compared with other frameworks for visualizing large maps, the Class Buffer model is more scalable over the three facets mentioned in the related work: data size, perceptual processing, and computation speed. The improved perceptual processing comes from the wide range of visualization techniques it can generate for multiclass density maps. The data size and speed scalability comes from the separation of computations between the back-end side and the front-end side. The back-end side needs to compute density maps, and the computation time is proportional to the number of points. Once this expensive computation is done, the wide range of visualizations offered by our model can be used to explore the data at interactive speed, regardless of the size of the original data. It can be about millions of points, billions, or any higher scale; the sheer amount is irrelevant to our model, and users can explore data using multiple visualizations with different trade-offs regarding the visual tasks supported. Other visualization libraries compute the aggregation and the visualization together, needing a time proportional to the number of points to generate a new visualization and require expensive round-trips from the back-end to the browser.

Abstract Rendering [13, 23] performs the binning operations in complex buffers that are configured early before their contents are computed. Once computed, most of the rendering has to be performed in the back-end as well because the composite structure at each bin is too complex to be sent transparently to a front-end. Each modification of the AR pipeline requires an expensive recomputation starting at the binning stage, requiring the handling of the whole dataset. In addition, similar to other scalable visualization systems [38], AR performs smoothing during the aggregation stage: each point value is modulated by a smoothing kernel that is added to the density array, an operation that slows down the computation of the density map by the size of the kernel (typically 4–16). By contrast, the Class Buffer model receives the raw unsmoothed density map and can apply a smoothing kernel to it at its first stage. According to Wickham [53], applying the smoothing to the binned data produces very similar results than applying it before binning, with a substantial performance improvement [51].

Compared with Wickham's BSS model [53], the Class Buffer model provides a richer set of visualization options, but less statistical oper-

Specification

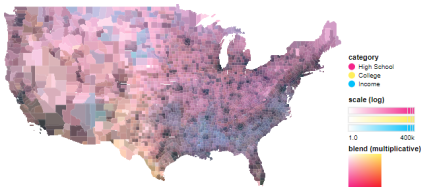
```
"smooth": {"radius": 1},
"rescale": {"type": "log"},
"compose": {"mix": "mean"},
"stroke": {
  "type": "topojson",
  "url": "us.json",
  "feature": "states",
  "color": "rgba(0, 0, 0, 0.3)"}
```



(a)

Specification

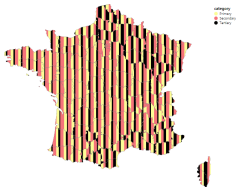
```
"rebin": {
  "type": "topojson",
  "url": "us.json",
  "feature": "counties"},
"rescale": { "type": "log" },
"compose": {
  "mix": "blend",
  "mixing": "multiplicative"}
```



(b)

Specification

```
"rebin": {
  "type": "topojson",
  "url": "franceD.json",
  "feature": "poly"},
"compose": {
  "mix": "propline",
  "size": 18,
  "widthprop": "percent"}
```



(c)

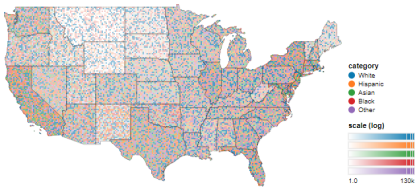
Specification

```
"rebin": // US rebinning
"compose": {
  "mix": "hatching", "size": 4,
  "widthprop": "percent",
  "colprop": true}
```

(d)

Specification

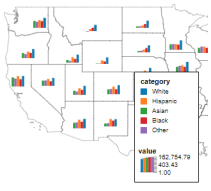
```
"rebin": // US rebinning
"compose": {
  "mix": "weavingrandom",
  "size": 2}
```



(e)

Specification

```
"rebin": // US rebinning
"compose": { "mix": "glyph",
  "glyphSpec": {
    "template": "bars",
    "width": 20, "height": 24}}
```



(f)

Fig. 6: Revisited examples of multiclass density maps (Fig. 2). The Class Buffer model can express various designs of multiclass density maps using a single declarative visualization grammar. Underlying data is synthetic.

Specification

```
"compose": {"mix": "max"},
"rescale": {"type": "equidepth"}
```

Specification

```
"compose": {"mix": "invmin"},
"rescale": {"type": "sqrt"},
"rebin": {"type": "square",
 "size": 2, "aggregation": "min"}
```

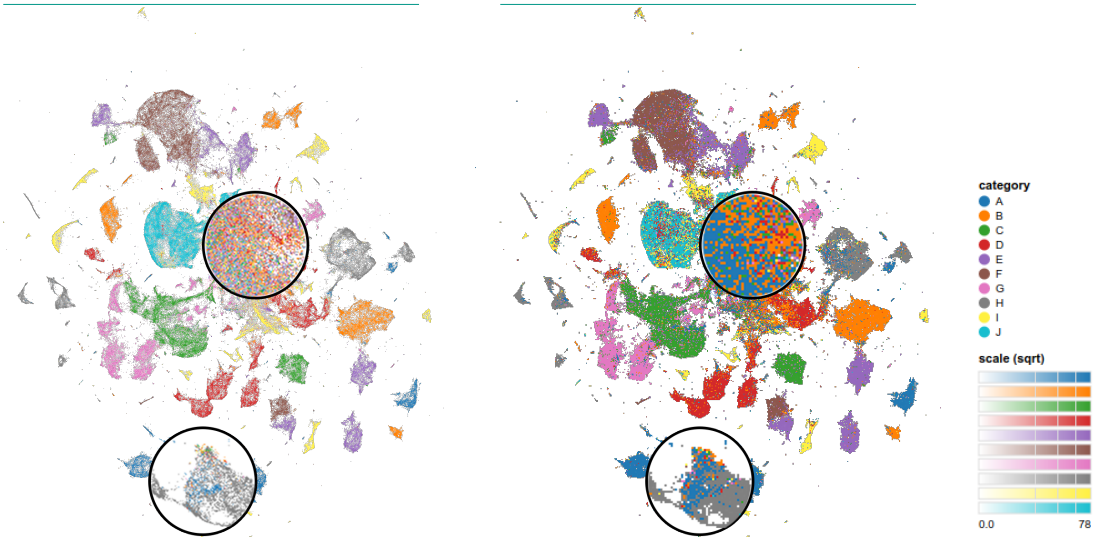


Fig. 7: The notMNIST dataset [11] projected using LargeVis [48]. On the left side, colors are blended to reveal global clusters. On the right side, the color of a class with the minimum density is chosen (`invmin`), revealing outliers and algorithm artifacts.

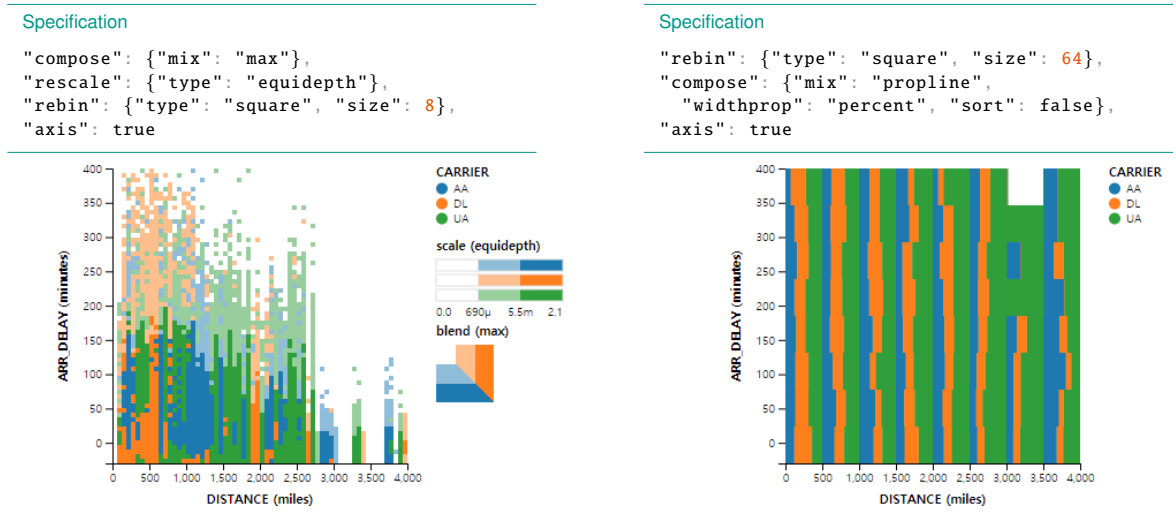


Fig. 8: Multiclass density maps showing the on-time performance of three carriers in the United States [37], augmented by axes. On the left side, the overall trend can be seen; most flights traveled shorter than 2,500 miles, and most long delays occurred from those flights. On the right side, proportional lines were used as stacked bar charts, enabling direct comparison between carriers.

ations at the prebinning stage. This is because the BSS model does not manage multiclass data. Composing multiple classes from density maps is understandable since densities are directly comparable, but with more complex statistics such as average, variance, or higher-order moments, the interpretations of the class combinations are challenging.

To be more specific on performance, we report times measured with the Datashader library [14], optimized to visualize large-scale aggregated data. We measured times to process the synthetic census data [50] containing longitude/latitude and five "races" for 300 million households in the US computed on a modern Linux laptop with 16GB of memory and using 2 cores. Loading from a local SSD using a compressed format took 11 s. Building five class buffers at a resolution of 900×525 took 1.26 s. Sending the data locally to the browser through a WebSocket took 0.22 s. Datashader mainly implements additive color mixing that ran in 0.088 s. Our implementation can render the class buffers using most configurations in around 0.1 s, up to 1 s when computing contour lines with a large blur radius. The rendering times are thus similar but our implementation saves network traffic—that can be slow when using a remote connection—and complements Datashader with a much richer set of rendering options.

6.2 Interactive Data Exploration

Our Class Buffer model uses data buffers (i.e., 2D histograms) as data sources, and this provides the model with an ability to abstract underlying computation of large-scale data. For example, we assumed that our dataset $T = (Q_1, Q_2, C)$ is fixed for simplicity, but in practice, the dataset (1) usually has more dimensions and (2) can dynamically change through user interaction such as filtering out rows. When the dataset is filtered by a new dimension, the data buffers should be invalidated and recomputed. In interactive exploration with multidimensional data, this would be a frequent case, and modern data structures [27, 28] have been proposed to speed up the recomputation. Our Class Buffer model naturally lends itself to working with those optimized data structures through transfer of abstract data buffers. In addition, the ProgressiVis toolkit [16] already transfers data buffers of progressively aggregated data to its front-end, which can be used with our model.

6.3 Limitations

While the Class Buffer model allows creating a wide variety of visualizations suited to complex tasks, it also needs additional data to be fully usable, such as landmarks, points of interest, and outliers; they should be combined at the rendering stage. Landmarks for maps include important locations and names, sometimes additional shapes to add context, such as rivers or points of interest. Landmarks for scatterplots and multidimensional projections include location of interesting points. For example, in publication data, highly cited publications or authors can be used as landmarks. Many tasks described in Sect. 2.1 mention identifying an individual point or comparing objects with groups. To support these tasks, a selection of important objects should be sent to the front-end for analysis.

The Class Buffer model can replicate several techniques used to visualize multiclass density maps, but it does not offer guidance on their best use. We need more experience to provide such guidelines, for example, through controlled experiments with different techniques. However, providing a usable implementation will enable the visualization community to start researching these maps, opening up a new area of scalable visualization.

7 Conclusion and Future Work

Despite the popularity of multiclass maps, little has been done regarding their design space and scalability. In this paper, we present the Class Buffer model to improve the expressiveness and scalability of multiclass density maps. Starting from the classic examples from cartography, we survey and abstract multiclass density map idioms into a unified Class Buffer model. Our model separates computation between the back-end side and the front-end side, allowing various map designs without repeating binning and aggregation. Through our declarative visualization grammar, users can explore the design space of multiclass density maps, finding the best design for achieving their goals.

For future work, we can extend our model with more multiclass density map idioms, in particular related to user interactions. For example, the user may want to pan and zoom a density map or manipulate color ramps on the legend through filtering and clamping. In addition, our prototype implementation is fast enough for modern browsers, but the performance can be improved using WebGL for high resolution screens or faster contour line computation. Finally, a body of research that investigated tasks and designs of conventional scatterplots exists [42], but we found such attempts are rare for multiclass density plots. We believe our reusable implementation will foster more research on multiclass density maps as an effective idiom of scalable visualization.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No. NRF-2016R1A2B2007153). Fig. 2c is from Semiology of Graphics: Diagrams, Networks, Maps by Jacques Bertin and reprinted by permission of the University of Wisconsin Press.

REFERENCES

[1] M. Attarha, C. M. Moore, and S. P. Vecera. The time-limited visual statistician. *Journal of Experimental Psychology: Human Perception and Performance*, 42(10):1497–1504, 2016. doi: 10.1037/xhp0000255

[2] M. Aupetit. Sanity Check for Class-coloring-based Evaluation of Dimension Reduction Techniques. In *Proceedings of the Fifth Workshop on Beyond Time and Errors: Novel Evaluation Methods for Visualization*, BELIV '14, pp. 134–141. ACM, New York, NY, USA, 2014. doi: 10.1145/2669557.2669578

[3] M. Aupetit and M. Sedlmair. SepMe: 2002 New visual separation measures. In *2016 IEEE Pacific Visualization Symposium (PacificVis)*, pp. 1–8, Apr. 2016. doi: 10.1109/PACIFICVIS.2016.7465244

[4] J. Bertin. *Semiology of Graphics: Diagrams, Networks, Maps*. Esri Press, Redlands, 2011.

[5] E. Bertini and G. Santucci. Give Chance a Chance: Modeling Density to Enhance Scatter Plot Quality through Random Data Sampling. *Information Visualization*, 5(2):95–110, 2006. doi: 10.1057/palgrave.ivs.9500122

[6] M. Bostock. TopoJSON. Online, 2017.

[7] M. Bostock, V. Ogievetsky, and J. Heer. D3 data-driven documents. *IEEE Trans. Vis. Comput. Graphics*, 17(12):2301–2309, Dec. 2011. doi: 10.1109/TVCG.2011.185

[8] M. Brehmer, M. Sedlmair, S. Ingram, and T. Munzner. Visualizing dimensionally-reduced data: Interviews with analysts and a characterization of task sequences. In *Proceedings of the Fifth Workshop on Beyond Time and Errors: Novel Evaluation Methods for Visualization*, BELIV '14, pp. 1–8. ACM, New York, NY, USA, 2014. doi: 10.1145/2669557.2669559

[9] C. Brewer and A. J. Campbell. Beyond graduated circles: varied point symbols for representing quantitative data on maps. *Cartographic Perspectives*, (29):6–25, 1998.

[10] W. C. Brinton. *Graphic presentation*. New York city, Brinton associates, 1939.

[11] Y. Bulatov. notMNIST dataset. `http://yaroslavvb.blogspot.fr/2011/09/notmnist-dataset.html`. Accessed: 2018-03-26.

[12] H. Chen, S. Engle, A. Joshi, E. D. Ragan, B. F. Yuksel, and L. Harrison. Using Animation to Alleviate Overdraw in Multiclass Scatterplot Matrices. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, New York, NY, USA, April 2018. doi: 10.1145/3173574.3173991

[13] J. Cottam, A. Lumsdaine, and P. Wang. Overplotting: Unified solutions under Abstract Rendering. In *2013 IEEE International Conference on Big Data*, pp. 9–16, Oct. 2013. doi: 10.1109/BigData.2013.6691712

[14] Datashader. `http://datashader.org/`. Accessed: 2018-03-31.

[15] J.-D. Fekete and C. Plaisant. Interactive Information Visualization of a Million Items. In I. Press, ed., *Proc. IEEE Symposium on Information Visualization 2002 (InfoVis 2002)*, pp. 117–124. IEEE, IEEE Press, Boston, United States, Oct. 2002. doi: 10.1109/INFVIS.2002.1173156

[16] J.-D. Fekete and R. Primet. Progressive Analytics: A Computation Paradigm for Exploratory Data Analysis. *ArXiv e-prints*, July 2016.

[17] A. J. Frère de Montizon. *Carte philosophique figurant la population de la France*. 1830.

[18] M. Gleicher. Considerations for visualizing comparison. *IEEE Trans. Vis. Comput. Graphics*, 24(1):413–423, Jan 2018. doi: 10.1109/TVCG.2017.2744199

[19] H. Hagh-Shenas, S. Kim, V. Interrante, and C. Healey. Weaving versus blending: a quantitative assessment of the information carrying capacities of two alternative methods for conveying multivariate data with color. *IEEE Trans. Vis. Comput. Graphics*, 13(6):1270–1277, Nov 2007. doi: 10.1109/TVCG.2007.70623

[20] R. L. Harris. *Information graphics: A comprehensive illustrated reference*. Oxford University Press, 2000.

[21] M. Harrower and C. A. Brewer. Colorbrewer.org: an online tool for selecting colour schemes for maps. *The Cartographic Journal*, 40(1):27–37, 2003.

[22] G. Hubacek. Transparency: Are the Richest Americans Also the Best Educated? America's Richest Counties and Best Educated Counties. Good Magazine, Online, Jan. 2011.

[23] P. W. Joseph A. Cottam, Andrew Lumsdaine. Abstract rendering: out-of-core rendering for information visualization. In *Proc. SPIE*, vol. 9017, p. 13, 2014. doi: 10.1117/12.2041200

[24] D. A. Keim, M. C. Hao, U. Dayal, H. Janetzko, and P. Bak. Generalized Scatter Plots. *Information Visualization*, 9(4):301–311, 2010. doi: 10.1057/ivs.2009.34

[25] J. Krygier. Map Symbols: Showing Multivariate Data with Texture. Online, 2008.

[26] D. Lamb. An automated displaced proportional circle map using delaunay triangulation and an algorithm for node overlap removal. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 52(4):364–370, 2017.

[27] L. Lins, J. T. Klosowski, and C. Scheidegger. Nanocubes for real-time exploration of spatiotemporal datasets. *IEEE Trans. Vis. Comput. Graphics*, 19(12):2456–2465, Dec 2013. doi: 10.1109/TVCG.2013.179

[28] Z. Liu, B. Jiang, and J. Heer. imMens: Real-time Visual Querying of Big Data. In *Proc. of the 15th Eurographics Conference on Visualization*, EuroVis '13, pp. 421–430. The Eurographs Association & John Wiley & Sons, Ltd., 2013. doi: 10.1111/cgf.12129

[29] R. Maciejewski. *Data Representations, Transformations, and Statistics for Visual Reasoning*. Morgan & Claypool, 2011.

[30] J. Matejka, F. Anderson, and G. Fitzmaurice. Dynamic opacity optimization for scatter plots. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pp. 2707–2710. ACM, 2015.

[31] A. Mayorga and M. Gleicher. Splatterplots: Overcoming Overdraw in Scatter Plots. *IEEE Trans. Vis. Comput. Graphics*, 19(9):1526–1538, sep 2013. doi: 10.1109/TVCG.2013.65

[32] L. Micallef, G. Palmas, A. Oulasvirta, and T. Weinkauf. Towards perceptual optimization of the visual design of scatterplots. *IEEE Trans. Vis. Comput. Graphics*, 23(6):1588–1599, 2017. doi: 10.1109/TVCG.2017.2674978

[33] J. R. Miller. Attribute Blocks: Visualizing Multiple Continuously Defined Attributes. *IEEE Computer Graphics and Applications*, 27(3):57–69, May 2007. doi: 10.1109/MCG.2007.54

[34] R. B. Miller. Response Time in Man-computer Conversational Transactions. In *Proc. of the Fall Joint Computer Conference, Part I*, pp. 267–277. ACM, 1968. doi: 10.1145/1476589.1476628

[35] T. Munzner. *Visualization Analysis and Design*. A.K. Peters visualization series. A K Peters, 2014.

[36] E. S. Nelson. Designing effective bivariate symbols: The influence of perceptual grouping processes. *Cartography and Geographic Information Science*, 27(4):261–278, 2000.

[37] B. of Transportation Statistics. On-Time Performance. Online, 2017.

[38] A. Perrot, R. Bourqui, N. Hanusse, F. Lalanne, and D. Auber. Large Interactive Visualization of Density Functions on Big Data Infrastructure. In *5th IEEE Symposium on Large Data Analysis and Visualization*. IEEE, Chicago, United States, Oct. 2015.

[39] N. Pezzotti, T. Höllt, B. P. Lelieveldt, E. Eisemann, and A. Vilanova. Hierarchical Stochastic Neighbor Embedding. *Computer Graphics Forum (Proc. of EuroVis)*, 35(3):21–30, June 2016.

[40] F. E. Pierce. Map No.2 of City of New-York, Showing the Distribution of the Principal Nationalities by Sanitary Districts. *Harper's Weekly*, 39(1985):60–61, January 1895.

[41] R. E. Roth. An Empirically-Derived Taxonomy of Interaction Primitives for Interactive Cartography and Geovisualization. *IEEE Trans. Vis. Comput. Graphics*, 19(12):2356–2365, Dec. 2013. doi: 10.1109/TVCG.2013.130

[42] A. Sarikaya and M. Gleicher. Scatterplots: Tasks, Data, and Designs. *IEEE Trans. Vis. Comput. Graphics*, 24(1), Jan. 2018. doi: 10.1109/TVCG.2017.2744184

[43] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-Lite: A Grammar of Interactive Graphics. *IEEE Trans. Vis. Comput. Graphics*, 2017.

[44] B. Shneiderman. Response Time and Display Rate in Human Performance with Computers. *ACM Comput. Surv.*, 16(3):265–285, Sept. 1984. doi: 10.1145/2514.2517

[45] A. Slingsby, J. Dykes, and J. Wood. Exploring uncertainty in geodemographics with interactive graphics. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2545–2554, 2011.

[46] D. B. Sparks. Isarithmic Maps of Public Opinion Data. Online, Oct. 2011.

[47] A. Spoerri. Infocrystal: A visual tool for information retrieval & management. In *Proceedings of the second international conference on Information and knowledge management*, pp. 11–20. ACM, 1993.

[48] J. Tang, J. Liu, M. Zhang, and Q. Mei. Visualizing Large-scale and High-dimensional Data. In *Proceedings of the 25th International Conference on World Wide Web*, pp. 287–297. International World Wide Web Conferences Steering Committee, 2016.

[49] B. E. Trumbo. A Theory for Coloring Bivariate Statistical Maps. *The American Statistician*, 35(4):220–226, 1981. doi: 10.1080/00031305.1981

.10479360

[50] RTI U.S. Synthetic Household Population^{TM}. `https://www.rti.org/impact/synthpop`. Accessed: 2018-03-20.

[51] M. P. Wand. Fast Computation of Multivariate Kernel Estimators. *Journal of Computational and Graphical Statistics*, 3(4):433–445, 1994. doi: 10.1080/10618600.1994.10474656

[52] C. Ware. Quantitative texton sequences for legible bivariate maps. *IEEE Trans. Vis. Comput. Graphics*, 15(6):1523–1530, Nov 2009. doi: 10.1109/TVCG.2009.175

[53] H. Wickham. Bin-summarise-smooth: a framework for visualising large data. unpublished, 2013.

[54] L. Wilkinson. *The Grammar of Graphics*. Springer-Verlag New York, Inc., 2005.